



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/511,877	10/19/2004	Anders Per Holmberg	P15178-US1	4220
27045	7590	03/23/2009		
ERICSSON INC. 6300 LEGACY DRIVE M/S EVR 1-C-11 PLANO, TX 75024				
EXAMINER				
TSAL SHENG JEN				
ART UNIT		PAPER NUMBER		
2186				
MAIL DATE		DELIVERY MODE		
03/23/2009		PAPER		

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary

Application No.

10/511,877

Applicant(s)

HOLMBERG ET AL.

Examiner

SHENG-JEN TSAI

Art Unit

2186

Period for Reply -- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

- 1) ☒ Responsive to communication(s) filed on 16 January 2009.
- 2a) ☒ This action is **FINAL**. 2b) ☐ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

- 4) ☒ Claim(s) 1, 2, 4-12, 15-18 and 20-23 is/are pending in the application.
- 4a) Of the above claim(s) _____ is/are withdrawn from consideration.
- 5) ☐ Claim(s) _____ is/are allowed.
- 6) ☒ Claim(s) 1, 2, 4-12, 15-18 and 20-23 is/are rejected.
- 7) ☐ Claim(s) _____ is/are objected to.
- 8) ☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☒ The drawing(s) filed on 19 October 2004 is/are: a) ☒ accepted or b) ☐ objected to by the Examiner.
- Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
- Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some * c) ☐ None of:
1. ☐ Certified copies of the priority documents have been received.
 2. ☐ Certified copies of the priority documents have been received in Application No. _____.
 3. ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

- 1) ☐ Notice of References Cited (PTO-892)
- 2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3) ☐ Information Disclosure Statement(s) (PTO/SB/08)
Paper No(s)/Mail Date _____
- 4) ☐ Interview Summary (PTO-413)
Paper No(s)/Mail Date _____
- 5) ☐ Notice of Informal Patent Application
- 6) ☐ Other: _____

DETAILED ACTION

1. This Office Action is taken in response to Applicants' Remarks filed on January 16, 2009 regarding application 10/511,877 filed on October 19, 2004.
2. Claims 1-2, 4-12, 14-18 and 20-23 are pending in the application under consideration.

3. ***Response to Remarks***

Applicants' remarks have been fully and carefully considered, with the Examiner's response set forth below.

(1) Applicants contend, regarding claim 1, that figure 3 of the Jacobson reference showing a one-way connection from the backing register file to the main memory, thus it fails to teach the limitation "for loading said special-purpose access register file from memory" as recited in the claim. The Examiner disagrees.

Regarding the limitation "for loading said special-purpose access register file from memory," Jacobson teaches [Backing Register File 300, being connected to both Register Files 308 and 310, may be used to hold, store, pre fetch, and temporarily buffer values in a way that will compliment the number of registers available locally to both the integer execution units and the floating point execution units ... Those that do will use Backing Register File 300 as just described, such as for temporary storage instead of using main memory, or to prefetch or preload values into Backing Register File 300 in preparation for execution (col. 4, lines 53-57 and col. 5, lines 1-5)] and [Communicating with main memory is a long process (many processor clock cycles), which could cause a stall state in one or more execution units as the values are read or

written between Register Files 308 or 310 and Main Memory 306. However, with Backing Register File 300 the chances of going into a stall state may be eliminated or at least minimized by using it to temporarily store results, or to hold prefetched values from Main Memory 306 in preparation for the instruction that needs those values ... (col. 5, lines 8-17)].

Significantly, Jacobson teaches that the Backing Register File is used to hold prefetched/preload values from Main Memory in order to eliminate a stall state. Note that holding prefetched/preload values from Main Memory is clearly an operation to load prefetched/preload values from the main memory to the Backing Register File.

Further, Jacobson teaches that "values are read or written between Register Files 308 or 310 and Main Memory 306." Thus the communication between Register Files 308 or 310 and Main Memory 306 are bidirectional. Figure 3 shows that the communication path between Register Files 308 or 310 and Main Memory 306 has to go through the Backing Register File 300, and that the connections between Register Files 308 or 310 and the Backing Register File 300 are bidirectional. Thus the connection between the Backing Register File 300 and Main Memory 306 has to also be directional in order to support bidirectional communication between Register Files 308 or 310 and Main Memory 306.

Thus, the Backing Register File of Jacobson indeed has a dedicated direct path connecting to the main memory that allows loading values into the Backing Register File. Hence the combination of Chatterjee and Jacobson teaches all the limitations recited in claim 1.

(2) Another iteration of claim analysis based on previously relied on references (Chatterjee et al., US 5,634,046 and Jacobson et al., US 7,206,925) has been made. Refer to the corresponding sections of the following claim analysis for details.

Claim Rejections - 35 USC § 103

4. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

5. Claims 1-2, 4-5, 7-12, 14, 17-18, 20-21 and 23 are rejected under 35 U.S.C. 103(a) as being unpatentable over Chatterjee et al. (US 5,634,046, hereinafter referred to as Chatterjee), and in view of Jacobson et al. (US 7,206,925, hereinafter referred to as Jacobson).

It is noted that, in the following claim analysis, those elements recited by the claims are presented in **bold font**.

As to claim 1, Chatterjee discloses **a computer system** [the stack pointer register in a computer ... (abstract)] **comprising:**
a dedicated special-purpose register file [the corresponding dedicated special-purpose register file includes the segment registers comprising CS (38), SS (39), DS (40), ES (41), FS (42) and GS (43), and the status registers comprising EFLAGS and EIP (figure 2, 44 and 45); In a typical computer, the CPU includes both general purpose and special purpose registers. General purpose registers are used for storing

operands, or pointers to operands in memory which are used for operations executed on the CPU. Special purpose registers generally store data related to limited purposes, such as for storing data relating to control, exceptions, memory management, and the like (column 1, 42-47)] **separate from other general register files of the computer system** [the corresponding general register file is the General Registers shown in figure 2, comprising EAX (30), EBX (31), ECX (32), EDX (33), EBP (34), ESI (35), EDI (36) and ESP (37)]; In a typical computer, the CPU includes both general purpose and special purpose registers. General purpose registers are used for storing operands, or pointers to operands in memory which are used for operations executed on the CPU. Special purpose registers generally store data related to limited purposes, such as for storing data relating to control, exceptions, memory management, and the like (column 1, 42-47)] **adapted solely for holding memory address calculation information received from memory** [The instruction pointer stored in the EIP register 45 is combined with a segment selector in the CS register 38 to obtain an address of the next instruction in the current code segment of the main memory 14 (column 5, lines 16-20); In transferring execution to the interrupt handler, the microprocessor 12 stores the current data from the instruction pointer register 45 and code segment register 38 (which constitute a "return address" for the program), as well as the flags register 44, with push operations. The microprocessor 12 also retrieves data relating to a starting address of the interrupt handler's first instruction for loading into the instruction pointer register 45 and code segment register 38 (column 5, line 66 to column 6, line 7); note that the contents of the special-purpose registers are

stored in the corresponding "stack segment" sections (figure 1, 20, 21 and 26; column 4, lines 18-32) of the main memory (figure 1, 14), and these information are retrieved from the main memory and loaded into the corresponding special-purpose registers to obtain the "return address" upon returning from an interrupt: In the Intel microprocessors, the stack is used to store a return address when switching execution between programs, such as when an interrupt occurs. Specifically, when the microprocessor receives an interrupt, the address of the instruction currently being executed is pushed onto the stack. The starting address of an interrupt handler (a program for servicing an interrupt) is then loaded into the instruction register from a predetermined location in main memory, and the microprocessor begins executing the interrupt handler. After completing execution of the interrupt handler, the microprocessor returns to the previously executing program by popping the address of the instruction where program execution left off from the stack, and loading that address into the instruction register (column 2, lines 8-21)], **said special-purpose register file having at least one dedicated interface for allowing efficient transfer of memory address calculation information in relation to said special-purpose register file** [figures 1 shows the interface between the register set (figure 1, 18 and figure 2, 18) and the main memory (figure 1, 14) via a bus (figure 1, 16); the "dedicated" aspect of the interface is taught by Jacobson, see below]; **wherein said at least one dedicated interface includes a dedicated direct path between said special-purpose register file and memory** [figures 1 shows a direct path interface between the register set (figure 1, 18 and figure 2, 18) and the main

memory (figure 1, 14) via a bus (figure 1, 16); the "dedicated" aspect of the interface is taught by Jacobson, see below] **for loading said special-purpose access register file from memory** [The instruction pointer stored in the EIP register 45 is combined with a segment selector in the CS register 38 to obtain an address of the next instruction in the current code segment of the main memory 14 (column 5, lines 16-20); In transferring execution to the interrupt handler, the microprocessor 12 stores the current data from the instruction pointer register 45 and code segment register 38 (which constitute a "return address" for the program), as well as the flags register 44, with push operations. The microprocessor 12 also retrieves data relating to a starting address of the interrupt handler's first instruction for loading into the instruction pointer register 45 and code segment register 38 (column 5, line 66 to column 6, line 7); note that the contents of the special-purpose registers are stored in the corresponding "stack segment" sections (figure 1, 20, 21 and 26; column 4, lines 18-32) of the main memory (figure 1, 14), and these information are retrieved from the main memory and loaded into the corresponding special-purpose registers to obtain the "return address" upon returning from an interrupt ; In the Intel microprocessors, the stack is used to store a return address when switching execution between programs, such as when an interrupt occurs. Specifically, when the microprocessor receives an interrupt, the address of the instruction currently being executed is pushed onto the stack. The starting address of an interrupt handler (a program for servicing an interrupt) is then loaded into the instruction register from a predetermined location in main memory, and the microprocessor begins executing the interrupt handler. After completing execution

of the interrupt handler, the microprocessor returns to the previously executing program by popping the address of the instruction where program execution left off from the stack, and loading that address into the instruction register (column 2, lines 8-21));

means for determining a memory address in response to memory address calculation information received from said special-purpose register file, thus enabling a corresponding memory access [The instruction pointer stored in the EIP register 45 is combined with a segment selector in the CS register 38 to obtain an address of the next instruction in the current code segment of the main memory 14 (column 5, lines 16-20); In transferring execution to the interrupt handler, the microprocessor 12 stores the current data from the instruction pointer register 45 and code segment register 38 (which constitute a "return address" for the program), as well as the flags register 44, with push operations. The microprocessor 12 also retrieves data relating to a starting address of the interrupt handler's first instruction for loading into the instruction pointer register 45 and code segment register 38 (column 5, line 66 to column 6, line 7); note that the contents of the special-purpose registers are stored in the corresponding "stack segment" sections (figure 1, 20, 21 and 26; column 4, lines 18-32) of the main memory (figure 1, 14), and these information are retrieved from the main memory and loaded into the corresponding special-purpose registers to obtain the "return address" upon returning from an interrupt: In the Intel microprocessors, the stack is used to store a return address when switching execution between programs, such as when an interrupt occurs. Specifically, when the microprocessor receives an

interrupt, the address of the instruction currently being executed is pushed onto the stack. The starting address of an interrupt handler (a program for servicing an interrupt) is then loaded into the instruction register from a predetermined location in main memory, and the microprocessor begins executing the interrupt handler. After completing execution of the interrupt handler, the microprocessor returns to the previously executing program by popping the address of the instruction where program execution left off from the stack, and loading that address into the instruction register (column 2, lines 8-21)].

Regarding claim 1, Chatterjee teaches said special-purpose register file having at least one interface for allowing efficient transfer of memory address calculation information in relation to said special-purpose register file [figures 1 shows a direct path interface between the register set (figure 1, 18 and figure 2, 18) and the main memory (figure 1, 14) via a bus (figure 1, 16)], but does not teach that this direct interface is a "dedicated interface" since the interface is also shared by the general-purpose registers.

However, Jacobson teaches in the invention "Backing Register File for Processor" **a computer system** [as shown in figure 1] **comprising:**
a dedicated special-purpose register file [the corresponding dedicated special-purpose register file comprises the Backing Register File as shown in figure 3, 300]
separate from other general register files of the computer system [the corresponding general register file comprises RF-1 and RF-2 as shown in figure 3, 308 and 310], **wherein said special-purpose register file having at least one dedicated**

interface for allowing efficient transfer of memory address calculation

information in relation to said special-purpose register file [figure 3 shows that there is a dedicated, direct path interface between the Backing Register File (300) and the main memory (306); Backing Register File 300, being connected to both Register Files 308 and 310, may be used to hold, store, pre fetch, and temporarily buffer values in a way that will compliment the number of registers available locally to both the integer execution units and the floating point execution units ... Those that do will use Backing Register File 300 as just described, such as for temporary storage instead of using main memory, or to prefetch or preload values into Backing Register File 300 in preparation for execution (col. 4, lines 53-57 and col. 5, lines 1-5); Communicating with main memory is a long process (many processor clock cycles), which could cause a stall state in one or more execution units as the values are read or written between Register Files 308 or 310 and Main Memory 306. However, with Backing Register File 300 the chances of going into a stall state may be eliminated or at least minimized by using it to temporarily store results, or to hold prefetched values from Main Memory 306 in preparation for the instruction that needs those values ... (col. 5, lines 8-17)].

Significantly, Jacobson teaches that the Backing Register File is used to hold prefetched/preload values from Main Memory in order to eliminate a stall state. Note that holding prefetched/preload values from Main Memory is clearly an operation to load prefetched/preload values from the main memory to the Backing Register File.

Further, Jacobson teaches that "values are read or written between Register Files 308 or 310 and Main Memory 306." Thus the communication between Register

Files 308 or 310 and Main Memory 306 are bidirectional. Figure 3 shows that the communication path between Register Files 308 or 310 and Main Memory 306 has to go through the Backing Register File 300, and that the connections between Register Files 308 or 310 and the Backing Register File 300 are bidirectional. Thus the connection between the Backing Register File 300 and Main Memory 306 has to also be directional in order to support bidirectional communication between Register Files 308 or 310 and Main Memory 306.

Jacobson also teaches that the motivation of having a dedicated, direct interface between the register file and the memory is to enhance the performance [It is expected that cost conscious implementations will not implement a connection between Main Memory 306 and Backing Register File 300, while implementations where performance has precedence over cost may make use of the extra speed available by having a more direct connection between Backing Register File 300 and Main Memory 306 (col. 4, lines 47-52)].

Therefore, it would have been obvious for one of ordinary skills in the art at the time of Applicants' invention to provide a dedicated interface between the special-purpose register and the memory, as demonstrated by Jacobson, and to incorporate it into the existing scheme disclosed by Chatterjee, in order to enhance the processing speed of data transfers and increase the performance of the computer system.

As to claim 2, Chatterjee in view of Jacobson teaches that **the computer system according to claim 1, further comprising means for effectuating a memory access based on the determined memory address** [Chatterjee: The instruction

pointer stored in the EIP register 45 is combined with a segment selector in the CS register 38 to obtain an address of the next instruction in the current code segment of the main memory 14 (column 5, lines 16-20); it is understood that once the address of the next instruction is obtained, the next instruction will be fetched from the corresponding location/address of the memory to continue with the execution of the instruction; The microprocessor 12 also retrieves data relating to a starting address of the interrupt handler's first instruction for loading into the instruction pointer register 45 and code segment register 38 (column 5, line 66 to column 6, line 7); In the Intel microprocessors, the stack is used to store a return address when switching execution between programs, such as when an interrupt occurs. Specifically, when the microprocessor receives an interrupt, the address of the instruction currently being executed is pushed onto the stack. The starting address of an interrupt handler (a program for servicing an interrupt) is then loaded into the instruction register from a predetermined location in main memory, and the microprocessor begins executing the interrupt handler. After completing execution of the interrupt handler, the microprocessor returns to the previously executing program by popping the address of the instruction where program execution left off from the stack, and loading that address into the instruction register (column 2, lines 8-21)].

As to claim 4, Chatterjee in view of Jacobson teaches that **the computer system according to claim 1, wherein said at least one dedicated interface comprises a dedicated interface between said special-purpose register file and said means for determining a memory address** [Chatterjee: figures 1 shows a direct path

interface between the register set (figure 1, 18 and figure 2, 18) and the main memory (figure 1, 14) via a bus (figure 1, 16); The instruction pointer stored in the EIP register 45 is combined with a segment selector in the CS register 38 to obtain an address of the next instruction in the current code segment of the main memory 14 (column 5, lines 16-20); In transferring execution to the interrupt handler, the microprocessor 12 stores the current data from the instruction pointer register 45 and code segment register 38 (which constitute a "return address" for the program), as well as the flags register 44, with push operations. The microprocessor 12 also retrieves data relating to a starting address of the interrupt handler's first instruction for loading into the instruction pointer register 45 and code segment register 38 (column 5, line 66 to column 6, line 7); note that the contents of the special-purpose registers are stored in the corresponding "stack segment" sections (figure 1, 20, 21 and 26; column 4, lines 18-32) of the main memory (figure 1, 14), and these information are retrieved from the main memory and loaded into the corresponding special-purpose registers to obtain the "return address" upon returning from an interrupt].

As to claim 5, Chatterjee in view of Jacobson teaches that **the computer system according to claim 1, wherein said at least one dedicated interface includes a dedicated data path adapted in width to said memory address calculation information** [Chatterjee: In the preferred embodiment of the invention, the application program can allocate either a 16-bit or a 32-bit segment of the memory 14 as the stack segment 26. If the stack segment 26 is allocated as a 16-bit segment, the microprocessor 12 stores only the contents of the lower 16-bit portion (referred to as

the SP register) 49 of the stack pointer register 37. When execution is later returned to the application program, only these lower 16-bits are restored. The upper 16-bits of the stack pointer register 37, however, are typically modified when execution is switched to the interrupt handler. Accordingly, when a 16-bit segment of the memory 14 is allocated as the stack segment 26, the application program should make use of only the lower 16-bit portion 49 of the stack pointer register 37 for storing data to avoid loss of the stored data. To allow use of the full 32-bits of the stack pointer register 37, the application program can allocate a 32-bit segment as the stack segment 26, such as by loading a segment selector for a 32-bit segment into the stack segment register 39 (column 7, lines 30-48)].

As to claim 7, Chatterjee in view of Jacobson teaches that **the computer system according to claim 1, wherein said means for determining a memory address comprises at least one functional processor unit** [Chatterjee: the microprocessor, figure 1, 12; The microprocessor 12 also retrieves data relating to a starting address of the interrupt handler's first instruction for loading into the instruction pointer register 45 and code segment register 38 (column 5, line 66 to column 6, line 7); The instruction pointer stored in the EIP register 45 is combined with a segment selector in the CS register 38 to obtain an address of the next instruction in the current code segment of the main memory 14 (column 5, lines 16-20); In the Intel microprocessors, the stack is used to store a return address when switching execution between programs, such as when an interrupt occurs. Specifically, when the microprocessor receives an interrupt, the address of the instruction currently being executed is pushed onto the stack. The

starting address of an interrupt handler (a program for servicing an interrupt) is then loaded into the instruction register from a predetermined location in main memory, and the microprocessor begins executing the interrupt handler. After completing execution of the interrupt handler, the microprocessor returns to the previously executing program by popping the address of the instruction where program execution left off from the stack, and loading that address into the instruction register (column 2, lines 8-21)).

As to claim 8, Chatterjee in view of Jacobson teaches that **the computer system according to claim 7, wherein a forwarding data path is arranged from an output bus associated with said at least one functional processor unit to an input bus associated with said at least one functional processor unit** [Chatterjee: as shown in figure 1, note that the bus is bidirectional as indicated by the arrows at the both ends].

As to claim 9, Chatterjee in view of Jacobson teaches that **the computer system according to claim 1, wherein said means for determining a memory address is operable for executing special-purpose instructions in order to determine said memory address** [Chatterjee: The Intel microprocessors respond to an instruction set which includes two stack operations, generally known as push and pop. The push operation stores additional data in a sequential order onto the stack, while the pop operation removes data from the stack in last-in-first-out ("LIFO") order (column 2, lines 2-7); The microprocessor inserts data into the stack segment with "push" operations. Push operations insert data into the stack segment 26 in a descending sequential order

starting at its highest address. The microprocessor removes data from the stack segment 26 in LIFO order using "pop" operations. A set of locations in the stack segment currently containing data is referred to as a "stack" 52 (column 5, lines 29-35); In the Intel microprocessors, the stack is used to store a return address when switching execution between programs, such as when an interrupt occurs. Specifically, when the microprocessor receives an interrupt, the address of the instruction currently being executed is pushed onto the stack. The starting address of an interrupt handler (a program for servicing an interrupt) is then loaded into the instruction register from a predetermined location in main memory, and the microprocessor begins executing the interrupt handler. After completing execution of the interrupt handler, the microprocessor returns to the previously executing program by popping the address of the instruction where program execution left off from the stack, and loading that address into the instruction register (column 2, lines 8-21)].

As to claim 10, Chatterjee in view of Jacobson teaches that **the computer system according to claim 1, further comprising means for executing special-purpose load instructions in order to load said memory address calculation information from said memory to said special-purpose register file** [Chatterjee: the corresponding special-purpose load instruction is the "pop" instruction; The Intel microprocessors respond to an instruction set which includes two stack operations, generally known as push and pop. The push operation stores additional data in a sequential order onto the stack, while the pop operation removes data from the stack in last-in-first-out ("LIFO") order (column 2, lines 2-7); The microprocessor inserts data

into the stack segment with "push" operations. Push operations insert data into the stack segment 26 in a descending sequential order starting at its highest address. The microprocessor removes data from the stack segment 26 in LIFO order using "pop" operations. A set of locations in the stack segment currently containing data is referred to as a "stack" 52 (column 5, lines 29-35); In the Intel microprocessors, the stack is used to store a return address when switching execution between programs, such as when an interrupt occurs. Specifically, when the microprocessor receives an interrupt, the address of the instruction currently being executed is pushed onto the stack. The starting address of an interrupt handler (a program for servicing an interrupt) is then loaded into the instruction register from a predetermined location in main memory, and the microprocessor begins executing the interrupt handler. After completing execution of the interrupt handler, the microprocessor returns to the previously executing program by popping the address of the instruction where program execution left off from the stack, and loading that address into the instruction register (column 2, lines 8-21)).

As to claim 11, Chatterjee in view of Jacobson teaches that **the computer system according to claim 10, wherein said means for executing special-purpose load instructions comprises at least one functional processor unit** [Chatterjee: The Intel microprocessors respond to an instruction set which includes two stack operations, generally known as push and pop. The push operation stores additional data in a sequential order onto the stack, while the pop operation removes data from the stack in last-in-first-out ("LIFO") order (column 2, lines 2-7); The microprocessor

inserts data into the stack segment with "push" operations. Push operations insert data into the stack segment 26 in a descending sequential order starting at its highest address. The microprocessor removes data from the stack segment 26 in LIFO order using "pop" operations. A set of locations in the stack segment currently containing data is referred to as a "stack" 52 (column 5, lines 29-35); In the Intel microprocessors, the stack is used to store a return address when switching execution between programs, such as when an interrupt occurs. Specifically, when the microprocessor receives an interrupt, the address of the instruction currently being executed is pushed onto the stack. The starting address of an interrupt handler (a program for servicing an interrupt) is then loaded into the instruction register from a predetermined location in main memory, and the microprocessor begins executing the interrupt handler. After completing execution of the interrupt handler, the microprocessor returns to the previously executing program by popping the address of the instruction where program execution left off from the stack, and loading that address into the instruction register (column 2, lines 8-21)].

As to claim 12, Chatterjee in view of Jacobson teaches that **the computer system according to claim 11, wherein a forwarding data path is arranged from said memory to an input wherein said memory address calculation information is in the form of implicit memory access information** [Chatterjee: figure 1 shows the data path between the microprocessor (12) and the memory (14); The Intel microprocessors respond to an instruction set which includes two stack operations, generally known as push and pop. The push operation stores additional data in a

sequential order onto the stack, while the pop operation removes data from the stack in last-in-first-out ("LIFO") order (column 2, lines 2-7); Note that both "push" and "pop" instructions are in the form of implicitly memory access information because the syntax of "push" and "pop" instructions do not explicitly specify any memory address information].

As to claim 14, Chatterjee in view of Jacobson teaches that **the computer system according to claim 23, wherein said implicit memory access information includes memory address translation information** [Chatterjee: The Intel microprocessors respond to an instruction set which includes two stack operations, generally known as push and pop. The push operation stores additional data in a sequential order onto the stack, while the pop operation removes data from the stack in last-in-first-out ("LIFO") order (column 2, lines 2-7); Note that both "push" and "pop" instructions are in the form of implicitly memory access information because the syntax of "push" and "pop" instructions do not explicitly specify any memory address information].

As to claim 17, it recites substantially the same limitations as in claim 1, and is rejected for the same reasons set forth in the analysis of claim 1. Refer to "As to claim 1" presented earlier in this Office Action for details.

As to claim 18, Chatterjee in view of Jacobson teaches that **the method according to claim 17, further comprising means for effectuating a memory access based on the determined memory address** [Chatterjee: The instruction pointer stored in the EIP register 45 is combined with a segment selector in the CS

register 38 to obtain an address of the next instruction in the current code segment of the main memory 14 (column 5, lines 16-20); it is understood that once the address of the next instruction is obtained, the next instruction will be fetched from the corresponding location/address of the memory to continue with the execution of the instruction; The microprocessor 12 also retrieves data relating to a starting address of the interrupt handler's first instruction for loading into the instruction pointer register 45 and code segment register 38 (column 5, line 66 to column 6, line 7); In the Intel microprocessors, the stack is used to store a return address when switching execution between programs, such as when an interrupt occurs. Specifically, when the microprocessor receives an interrupt, the address of the instruction currently being executed is pushed onto the stack. The starting address of an interrupt handler (a program for servicing an interrupt) is then loaded into the instruction register from a predetermined location in main memory, and the microprocessor begins executing the interrupt handler. After completing execution of the interrupt handler, the microprocessor returns to the previously executing program by popping the address of the instruction where program execution left off from the stack, and loading that address into the instruction register (column 2, lines 8-21)].

As to claim 20, Chatterjee in view of Jacobson teaches that **the method according to claim 17, wherein said at least one dedicated interface comprises a dedicated interface between said special-purpose register file and said means for determining a memory address** [Jacobson: figure 3 shows that there is a dedicated, direct path interface between the Backing Register File (300) and the main

memory (306); Backing Register File 300 may also be connected to Main Memory 306 through Connection 304. As will be readily understood by those of ordinary skill in the art and with the benefit of the present disclosure, Main Memory 306 is not located on the processor chip and Connection 304 is comprised of a series of connections and interfaces both on and off the chip as more fully described in FIG. 1, with Main Memory 306 being of conventional and well known design. It is expected that cost conscious implementations will not implement a connection between Main Memory 306 and Backing Register File 300, while implementations where performance has precedence over cost may make use of the extra speed available by having a more direct connection between Backing Register File 300 and Main Memory 306 (col. 4, lines 39-52); Chatterjee: figures 1 shows a direct path interface between the register set (figure 1, 18 and figure 2, 18) and the main memory (figure 1, 14) via a bus (figure 1, 16); The instruction pointer stored in the EIP register 45 is combined with a segment selector in the CS register 38 to obtain an address of the next instruction in the current code segment of the main memory 14 (column 5, lines 16-20); In transferring execution to the interrupt handler, the microprocessor 12 stores the current data from the instruction pointer register 45 and code segment register 38 (which constitute a "return address" for the program), as well as the flags register 44, with push operations. The microprocessor 12 also retrieves data relating to a starting address of the interrupt handler's first instruction for loading into the instruction pointer register 45 and code segment register 38 (column 5, line 66 to column 6, line 7); note that the contents of the special-purpose registers are stored in the corresponding "stack segment" sections

(figure 1, 20, 21 and 26; column 4, lines 18-32) of the main memory (figure 1, 14), and these information are retrieved from the main memory and loaded into the corresponding special-purpose registers to obtain the "return address" upon returning from an interrupt].

As to claim 21, Chatterjee in view of Jacobson teaches that **the method according to claim 17, wherein said at least one dedicated interface includes a dedicated data path adapted in width to said memory address calculation information** [Jacobson: figure 3 shows that there is a dedicated, direct path interface between the Backing Register File (300) and the main memory (306); Backing Register File 300 may also be connected to Main Memory 306 through Connection 304. As will be readily understood by those of ordinary skill in the art and with the benefit of the present disclosure, Main Memory 306 is not located on the processor chip and Connection 304 is comprised of a series of connections and interfaces both on and off the chip as more fully described in FIG. 1, with Main Memory 306 being of conventional and well known design. It is expected that cost conscious implementations will not implement a connection between Main Memory 306 and Backing Register File 300, while implementations where performance has precedence over cost may make use of the extra speed available by having a more direct connection between Backing Register File 300 and Main Memory 306 (col. 4, lines 39-52); Chatterjee: In the preferred embodiment of the invention, the application program can allocate either a 16-bit or a 32-bit segment of the memory 14 as the stack segment 26. If the stack segment 26 is allocated as a 16-bit segment, the microprocessor 12 stores only the

contents of the lower 16-bit portion (referred to as the SP register) 49 of the stack pointer register 37. When execution is later returned to the application program, only these lower 16-bits are restored. The upper 16-bits of the stack pointer register 37, however, are typically modified when execution is switched to the interrupt handler. Accordingly, when a 16-bit segment of the memory 14 is allocated as the stack segment 26, the application program should make use of only the lower 16-bit portion 49 of the stack pointer register 37 for storing data to avoid loss of the stored data. To allow use of the full 32-bits of the stack pointer register 37, the application program can allocate a 32-bit segment as the stack segment 26, such as by loading a segment selector for a 32-bit segment into the stack segment register 39 (column 7, lines 30-48)].

As to claim 23, Chatterjee in view of Jacobson teaches that **the computer system according to claim 11, wherein said memory address calculation information is in the form of implicit memory access information** [Chatterjee: figure 1 shows the data path between the microprocessor (12) and the memory (14); The Intel microprocessors respond to an instruction set which includes two stack operations, generally known as push and pop. The push operation stores additional data in a sequential order onto the stack, while the pop operation removes data from the stack in last-in-first-out ("LIFO") order (column 2, lines 2-7); Note that both "push" and "pop" instructions are in the form of implicitly memory access information because the syntax of "push" and "pop" instructions do not explicitly specify any memory address information].

6. Claim 6, 15-16 and 22 are rejected under 35 U.S.C. 103(a) as being unpatentable over Chatterjee et al. (US 5,634,046, hereinafter referred to as Chatterjee), in view of Jacobson et al. (US 5,854,939, hereinafter referred to as Jacobson), and further in view of Aikawa et al. (US 5,371,865, hereinafter referred to as Aikawa).

Regarding claim 6, Chatterjee in view of Jacobson teaches that typical computers may also have an intermediate or cache memory which provides a quicker average access time to data stored in main memory [Chatterjee: column 1, lines 20-22], but does not explicitly teaching using a dedicated cache adapted for memory address calculation information.

However, Aikawa teaches that **said memory comprises a dedicated cache adapted for said memory address calculation information** [A computer having a main memory for storing a plurality of data, a cache memory for temporarily storing a portion of the plurality of data, a processor for accessing data stored in the cache memory and processing the data according to instructions. The processor has an access instruction combined with a preload instruction, and an access instruction only for accessing data, and includes indicator circuitry for indicating a preload condition to the cache memory when the processor accesses data from the cache memory according to the access instruction combined with the preload instruction. The cache memory preloads data to be accessed next by the processor from the main memory when the processor indicates the preload condition (abstract)].

It is well known in the art that using a cache memory reduces memory access latency and increases the throughput, as Chatterjee in view of Jacobson teaches that typical computers may also have an intermediate or cache memory which provides a quicker average access time to data stored in main memory [Chatterjee: column 1, lines 20-22].

Therefore, it would have been obvious for one of ordinary skills in the art at the time of Applicants' invention to use a cache memory adapted for memory address calculation information, as alluded by Chatterjee in view of Jacobson and explicitly demonstrated by Aikawa, and to incorporate it into the existing apparatus disclosed by Chatterjee in view of Jacobson, to reduce memory access latency and to increase the throughput.

As to claim 15, Chatterjee discloses **a computer system** [the stack pointer register in a computer ... (abstract)] **comprising:**

A dedicated cache adapted for holding memory access information [figure 1 shows a main memory (14) having "stack" segments (21 and 26) holding "return address," or address for the next instruction; the dedicated cache aspect is taught by Aikawa, see below];

a dedicated special-purpose register file [the corresponding dedicated special-purpose register file includes the segment registers comprising CS (38), SS (39), DS (40), ES (41), FS (42) and GS (43), and the status registers comprising EFLAGS and EIP (figure 2, 44 and 45); In a typical computer, the CPU includes both general purpose and special purpose registers. General purpose registers are used for storing

operands, or pointers to operands in memory which are used for operations executed on the CPU. Special purpose registers generally store data related to limited purposes, such as for storing data relating to control, exceptions, memory management, and the like (column 1, 42-47)] **separate from other general register files of the computer system** [the corresponding general register file is the General Registers shown in figure 2, comprising EAX (30), EBX (31), ECX (32), EDX (33), EBP (34), ESI (35), EDI (36) and ESP (37); In a typical computer, the CPU includes both general purpose and special purpose registers. General purpose registers are used for storing operands, or pointers to operands in memory which are used for operations executed on the CPU. Special purpose registers generally store data related to limited purposes, such as for storing data relating to control, exceptions, memory management, and the like (column 1, 42-47)] **and adapted solely for holding memory address calculation information received from said dedicated cache over a first dedicated interface** [The instruction pointer stored in the EIP register 45 is combined with a segment selector in the CS register 38 to obtain an address of the next instruction in the current code segment of the main memory 14 (column 5, lines 16-20); In transferring execution to the interrupt handler, the microprocessor 12 stores the current data from the instruction pointer register 45 and code segment register 38 (which constitute a "return address" for the program), as well as the flags register 44, with push operations. The microprocessor 12 also retrieves data relating to a starting address of the interrupt handler's first instruction for loading into the instruction pointer register 45 and code segment register 38 (column 5, line 66 to column 6, line 7); note

that the contents of the special-purpose registers are stored in the corresponding "stack segment" sections (figure 1, 20, 21 and 26; column 4, lines 18-32) of the main memory (figure 1, 14), and these information are retrieved from the main memory and loaded into the corresponding special-purpose registers to obtain the "return address" upon returning from an interrupt: In the Intel microprocessors, the stack is used to store a return address when switching execution between programs, such as when an interrupt occurs. Specifically, when the microprocessor receives an interrupt, the address of the instruction currently being executed is pushed onto the stack. The starting address of an interrupt handler (a program for servicing an interrupt) is then loaded into the instruction register from a predetermined location in main memory, and the microprocessor begins executing the interrupt handler. After completing execution of the interrupt handler, the microprocessor returns to the previously executing program by popping the address of the instruction where program execution left off from the stack, and loading that address into the instruction register (column 2, lines 8-21); figures 1 shows the interface between the register set (figure 1, 18 and figure 2, 18) and the main memory (figure 1, 14) via a bus (figure 1, 16));

wherein said first dedicated interface includes a dedicated direct path between said special-purpose register file and the dedicated cache [figures 1 shows a direct path interface between the register set (figure 1, 18 and figure 2, 18) and the main memory (figure 1, 14) via a bus (figure 1, 16)] **for loading said special-purpose access register file from the dedicated cache** [The instruction pointer stored in the EIP register 45 is combined with a segment selector in the CS register 38 to obtain an

address of the next instruction in the current code segment of the main memory 14 (column 5, lines 16-20); In transferring execution to the interrupt handler, the microprocessor 12 stores the current data from the instruction pointer register 45 and code segment register 38 (which constitute a "return address" for the program), as well as the flags register 44, with push operations. The microprocessor 12 also retrieves data relating to a starting address of the interrupt handler's first instruction for loading into the instruction pointer register 45 and code segment register 38 (column 5, line 66 to column 6, line 7); note that the contents of the special-purpose registers are stored in the corresponding "stack segment" sections (figure 1, 20, 21 and 26; column 4, lines 18-32) of the main memory (figure 1, 14), and these information are retrieved from the main memory and loaded into the corresponding special-purpose registers to obtain the "return address" upon returning from an interrupt ; In the Intel microprocessors, the stack is used to store a return address when switching execution between programs, such as when an interrupt occurs. Specifically, when the microprocessor receives an interrupt, the address of the instruction currently being executed is pushed onto the stack. The starting address of an interrupt handler (a program for servicing an interrupt) is then loaded into the instruction register from a predetermined location in main memory, and the microprocessor begins executing the interrupt handler. After completing execution of the interrupt handler, the microprocessor returns to the previously executing program by popping the address of the instruction where program execution left off from the stack, and loading that address into the instruction register (column 2, lines 8-21));

means for determining a memory address in response to memory access information received from said special-purpose register file over a second dedicated interface, and means for effectuating a corresponding memory access based on the determined memory address [the corresponding second dedicated interface is the internal interface between the microprocessor and the segment registers (figure 2, 38~43) and the status registers (figure 2, 44-45), because the microprocessor has to "write to" and "read from" these special-purpose registers to obtain the address of the next instruction; The instruction pointer stored in the EIP register 45 is combined with a segment selector in the CS register 38 to obtain an address of the next instruction in the current code segment of the main memory 14 (column 5, lines 16-20); In transferring execution to the interrupt handler, the microprocessor 12 stores the current data from the instruction pointer register 45 and code segment register 38 (which constitute a "return address" for the program), as well as the flags register 44, with push operations. The microprocessor 12 also retrieves data relating to a starting address of the interrupt handler's first instruction for loading into the instruction pointer register 45 and code segment register 38 (column 5, line 66 to column 6, line 7); note that the contents of the special-purpose registers are stored in the corresponding "stack segment" sections (figure 1, 20, 21 and 26; column 4, lines 18-32) of the main memory (figure 1, 14), and these information are retrieved from the main memory and loaded into the corresponding special-purpose registers to obtain the "return address" upon returning from an interrupt: In the Intel microprocessors, the stack is used to store a return address when switching execution between programs,

such as when an interrupt occurs. Specifically, when the microprocessor receives an interrupt, the address of the instruction currently being executed is pushed onto the stack. The starting address of an interrupt handler (a program for servicing an interrupt) is then loaded into the instruction register from a predetermined location in main memory, and the microprocessor begins executing the interrupt handler. After completing execution of the interrupt handler, the microprocessor returns to the previously executing program by popping the address of the instruction where program execution left off from the stack, and loading that address into the instruction register (column 2, lines 8-21)].

Regarding claim 15, Chatterjee teaches said special-purpose register file having at least one interface for allowing efficient transfer of memory address calculation information in relation to said special-purpose register file [figures 1 shows a direct path interface between the register set (figure 1, 18 and figure 2, 18) and the main memory (figure 1, 14) via a bus (figure 1, 16)], but does not teach that this direct interface is a "dedicated interface" since the interface is also shared by the general-purpose registers.

However, Jacobson teaches in the invention "Backing Register File for Processor" **a computer system** [as shown in figure 1] **comprising:**
a dedicated special-purpose register file [the corresponding dedicated special-purpose register file comprises the Backing Register File as shown in figure 3, 300]
separate from other general register files of the computer system [the corresponding general register file comprises RF-1 and RF-2 as shown in figure 3, 308]

and 310], **wherein said special-purpose register file having at least one dedicated interface for allowing efficient transfer of memory address calculation information in relation to said special-purpose register file** [figure 3 shows that there is a dedicated, direct path interface between the Backing Register File (300) and the main memory (306); Backing Register File 300, being connected to both Register Files 308 and 310, may be used to hold, store, pre fetch, and temporarily buffer values in a way that will compliment the number of registers available locally to both the integer execution units and the floating point execution units ... Those that do will use Backing Register File 300 as just described, such as for temporary storage instead of using main memory, or to prefetch or preload values into Backing Register File 300 in preparation for execution (col. 4, lines 53-57 and col. 5, lines 1-5); Communicating with main memory is a long process (many processor clock cycles), which could cause a stall state in one or more execution units as the values are read or written between Register Files 308 or 310 and Main Memory 306. However, with Backing Register File 300 the chances of going into a stall state may be eliminated or at least minimized by using it to temporarily store results, or to hold prefetched values from Main Memory 306 in preparation for the instruction that needs those values ... (col. 5, lines 8-17)].

Significantly, Jacobson teaches that the Backing Register File is used to hold prefetched/preload values from Main Memory in order to eliminate a stall state. Note that holding prefetched/preload values from Main Memory is clearly an operation to load prefetched/preload values from the main memory to the Backing Register File.

Further, Jacobson teaches that "values are read or written between Register Files 308 or 310 and Main Memory 306." Thus the communication between Register Files 308 or 310 and Main Memory 306 are bidirectional. Figure 3 shows that the communication path between Register Files 308 or 310 and Main Memory 306 has to go through the Backing Register File 300, and that the connections between Register Files 308 or 310 and the Backing Register File 300 are bidirectional. Thus the connection between the Backing Register File 300 and Main Memory 306 has to also be directional in order to support bidirectional communication between Register Files 308 or 310 and Main Memory 306.

Jacobson also teaches that the motivation of having a dedicated, direct interface between the register file and the memory is to enhance the performance [It is expected that cost conscious implementations will not implement a connection between Main Memory 306 and Backing Register File 300, while implementations where performance has precedence over cost may make use of the extra speed available by having a more direct connection between Backing Register File 300 and Main Memory 306 (col. 4, lines 47-52)].

Therefore, it would have been obvious for one of ordinary skills in the art at the time of Applicants' invention to provide a dedicated interface between the special-purpose register and the memory, as demonstrated by Jacobson, and to incorporate it into the existing scheme disclosed by Chatterjee, in order to enhances the processing speed of data transfers and increases the performance of the computer system.

Regarding claim 15, Chatterjee in view of Jacobson teaches that typical computers may also have an intermediate or cache memory which provides a quicker average access time to data stored in main memory [Chatterjee: column 1, lines 20-22], but does not explicitly teaching using a dedicated cache adapted for memory address calculation information.

However, Aikawa teaches that **said memory comprises a dedicated cache adapted for said memory address calculation information** [A computer having a main memory for storing a plurality of data, a cache memory for temporarily storing a portion of the plurality of data, a processor for accessing data stored in the cache memory and processing the data according to instructions. The processor has an access instruction combined with a preload instruction, and an access instruction only for accessing data, and includes indicator circuitry for indicating a preload condition to the cache memory when the processor accesses data from the cache memory according to the access instruction combined with the preload instruction. The cache memory preloads data to be accessed next by the processor from the main memory when the processor indicates the preload condition (abstract)].

It is well known in the art that using a cache memory reduces memory access latency and increases the throughput, as Chatterjee in view of Jacobson teaches that typical computers may also have an intermediate or cache memory which provides a quicker average access time to data stored in main memory [Chatterjee: column 1, lines 20-22].

Therefore, it would have been obvious for one of ordinary skills in the art at the time of Applicants' invention to use a cache memory adapted for memory address calculation information, as alluded by Chatterjee in view of Jacobson and explicitly demonstrated by Aikawa, and to incorporate it into the existing apparatus disclosed by Chatterjee in view of Jacobson, to reduce memory access latency and to increase the throughput.

As to claim 16, Chatterjee in view of Jacobson & Aikawa teaches that **the computer system according to claim 15, wherein said first and second dedicated interfaces are adapted in width to said memory address calculation information** [Chatterjee: In the preferred embodiment of the invention, the application program can allocate either a 16-bit or a 32-bit segment of the memory 14 as the stack segment 26. If the stack segment 26 is allocated as a 16-bit segment, the microprocessor 12 stores only the contents of the lower 16-bit portion (referred to as the SP register) 49 of the stack pointer register 37. When execution is later returned to the application program, only these lower 16-bits are restored. The upper 16-bits of the stack pointer register 37, however, are typically modified when execution is switched to the interrupt handler. Accordingly, when a 16-bit segment of the memory 14 is allocated as the stack segment 26, the application program should make use of only the lower 16-bit portion 49 of the stack pointer register 37 for storing data to avoid loss of the stored data. To allow use of the full 32-bits of the stack pointer register 37, the application program can allocate a 32-bit segment as the stack segment 26, such as by loading a segment

selector for a 32-bit segment into the stack segment register 39 (column 7, lines 30-48)).

As to claim 22, Chatterjee in view of Jacobson & Aikawa teaches **the method according to claim 17, further comprising the step of utilizing a dedicated cache adapted for said memory address calculation information** [Aikawa: A computer having a main memory for storing a plurality of data, a cache memory for temporarily storing a portion of the plurality of data, a processor for accessing data stored in the cache memory and processing the data according to instructions. The processor has an access instruction combined with a preload instruction, and an access instruction only for accessing data, and includes indicator circuitry for indicating a preload condition to the cache memory when the processor accesses data from the cache memory according to the access instruction combined with the preload instruction. The cache memory preloads data to be accessed next by the processor from the main memory when the processor indicates the preload condition (abstract)].

7. Related Prior Art of Record

The following list of prior art is considered to be pertinent to applicant's invention, but not relied upon for claim analysis in this Office Action.

- Koino, (US 5,491,826), "Microprocessor Having Register Bank and Using a General Purpose Register as a Stack Pointer."
- Cohen et al., (US 5,115,506), "Method and Apparatus for Preventing Recursion Jeopardy."

- Shibasaki et al., (US 4,334,269), "Data Processing System Having an Integrated Stack and Register Machine Architecture."
- Morris et al., (US 6,631,460), "Advanced Load Address Table Entry Invalidation Based on Register Address Wraparound."
- Baror et al., (US 4,926,323), "Streamlined Instruction processor."
- Henry et al., (US 6,862,670), "Tagged Address Stack and Microprocessor Using Same."

Conclusion

8. Claims 1-2, 4-12, 14-18 and 20-23 are rejected as explained above.

9. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

10. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Sheng-Jen Tsai whose telephone number is 571-272-4244. The examiner can normally be reached on 8:30 - 5:00.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Matthew Kim can be reached on 571-272-4182. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300. Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/Sheng-Jen Tsai/

Primary Examiner, Art Unit 2186

March 17, 2009